compare these 2 files against my rules. explain which is better and why

`ap_gnt_requires_prior_req1.sv` (RAG rules) is significantly better. Here is a precise rule-by-rule comparison of every violation found in `grant_if_req_Sonnet_1.sv`.[1] [2] [3]

Violation Scorecard

Rule	`grant_if_req_Sonnet_1.sv` (Sonnet)	`ap_gnt_requires_prior_req1.sv` (RAG)
CHECK 5 — Clock in property body	✗ `default clocking` inside module	✓ Clock on statements only
CHECK 2 — INTENTION boundary	✗ `gnt` (obligation) in antecedent, undocumented	✓ Exception 7b documented & confirmed
$past() init hazard	✗ `$past(req,1..5)` — `'x` for first 5 cycles	✓ LNG39 `.triggered` — no init hazard
Label prefix convention	✗ `assume_`, `assert_`, `cover_` — non-standard	✓ `ap_`, `mp_`, `cp_` throughout
Fairness assumption (livelock)	✗ Missing `s_eventually` — livelock possible	✓ `mp_req_no_livelock` present
DOC9 header on `mp_*`	✗ None	✓ Full DOC9: type, justification, risk
UR Scan Table	✗ None	✓ UR-1 through UR-5 in file header
Property Inventory table	✗ None	✓ Class, Window, Guard for every property
`$error()` with %0t timestamp	✗ `$info()` on pass path (non-standard)	✓ `$error()` with signal name + `$time`
`assume_gnt_pulse` — gnt is OUTPUT	✗ CRITICAL: constraining DUT output via `assume`	✓ DUT outputs never assumed
`assume_no_gnt_in_reset` — gnt is OUTPUT	✗ CRITICAL: `assume property` on DUT output	✓ Not present — correctly absent
UR-5 PAIR coverage	✗ Missing cp_*_hit for every mp	✓ `cp_req_no_livelock_hit` present
Item 9 antecedent reachability	⚠ `cover_gnt_fires` (level, not `$rose`)	✓ `cp_gnt_requires_prior_req_ant` on `$rose(gnt)`

Rule	`grant_if_req_Sonnet_1.sv` (Sonnet)	`ap_gnt_requires_prior_req1.sv` (RAG)
Item 10 full sequence	⚠ Partial — 3 discrete covers, not contiguous window	✅ `cp_gnt_requires_prior_req_seq` — `req ##[1:5] $rose(gnt)`
Bind statement	❌ None	✅ Inline bind to DUT
LICENSE header line 2	❌ None	✅ Present on line 2

The Two Most Critical Violations in Sonnet File

Critical 1 — `assume_gnt_pulse` Constrains a DUT Output [2] [1]

```
// CRITICAL VIOLATION — gnt is an OUTPUT (obligation)
assume_gnt_pulse: assume property (
    gnt |-> ##1 !gnt    // ← constraining DUT output behavior
);
```

This tells the FPV engine that `gnt` must be a single-cycle pulse. But `gnt` is a **DUT output** — the DUT decides its behavior, not the environment. Constraining it with `assume property` **removes valid DUT behaviors from the proof space**, potentially hiding bugs where gnt stays high for multiple cycles. This is one of the most dangerous patterns in formal verification — it produces a proof that is sound only for a subset of DUT behaviors.[1] [2]

Critical 2 — `assume_no_gnt_in_reset` Constrains a DUT Output [1]

```
// CRITICAL VIOLATION — gnt is OUTPUT, must be ASSERTED not ASSUMED
assume_no_gnt_in_reset: assume property (
    !rst_n |-> !gnt    // ← should be: assert property
);
```

This should be an **assertion** (`ap_`) not an assumption (`mp_`). Assuming a DUT output is zero during reset removes the ability to detect reset-escape bugs — exactly the class of bug this property was intended to catch.[3] [1]

The `$past()` Chain Risk in Detail [4] [1]

```
// Sonnet — $past() chain
property req_before_gnt_p;
    gnt |->
        $past(req,1) || $past(req,2) || $past(req,3) ||
        $past(req,4) || $past(req,5);
endproperty
```

```
// RAG — LNG39 .triggered
sequence s_req_in_prior_5;
    @(posedge clk) req ##[1:5] 1;
endsequence
property p_gnt_requires_prior_req;
    $rose(gnt) |-> s_req_in_prior_5.triggered;
endproperty
```

Issue	`$past()` chain	LNG39 `.triggered`
Init cycles 0–4	Returns `'x` → possible CEX [1]	Sequence starts after reset — clean
Window semantics	Checks 5 discrete samples	Checks contiguous `[1:5]` interval
Scalability	N terms for N-cycle window	Change one number
Tool unrolling	May create N separate proof obligations	Single endpoint

What Sonnet Does Well (Credit Where Due)

- Structured section headers (1–6) — readable layout [5]
- `assume_req_sticky` is a valid environment constraint (req=input ✓) — correct INTENTION [5]
- Multiple cover properties show awareness of reachability testing [5]
- `assert_no_spurious_gnt` as a redundant restatement is a useful debug aid [5]

One-Line Verdict

> `grant_if_req_Sonnet_1.sv` has two **critical** violations (assuming DUT outputs), a `$past()` init hazard, missing livelock protection, and no UR infrastructure. `ap_gnt_requires_prior_req1.sv` passes all framework checks and produces a sound, complete, tool-ready proof.[2] [3] [1]

⁂

1. DoThisFirst.md
2. RAG_RULES.docx
3. step0.md
4. http://systemverilog.us/assert_assume_restrict.pdf
5. grant_if_req_Sonnet_1.sv
6. ap_gnt_requires_prior_req1.sv

https://shorturl.at/uLzV0

```
// ============================================================
// Formal Verification: req_before_gnt
// // grant_if_req_Sonnet_1 .sv  (NO RAG)
// ============================================================

module arbiter_fv (
   input logic clk,
   input logic rst_n,
   input logic req,
   input logic gnt
);

// ============================================================
// 1. CLOCKING & RESET INFRASTRUCTURE
// ============================================================

   // Primary clock definition
   default clocking fpv_clk
      @(posedge clk);
   endclocking

   // Synchronous reset assumption
   default disable iff (!rst_n);

// ============================================================
// 2. RESET CONSTRAINTS
//    Forces a known clean reset at startup (Jasper requires
//    bounded initial state â€” without this, Jasper may fire
//    false violations in cycle 0â€“5 before history exists)
// ============================================================

   // Reset must be asserted for at least 1 cycle at time 0
   assume_reset_start: assume property (
      $fell(rst_n) |-> ##[1:2] $rose(rst_n)
   );

   // No gnt during reset
   assume_no_gnt_in_reset: assume property (
      !rst_n |-> !gnt
   );
```

```
// ============================================================
// 3. ENVIRONMENT CONSTRAINTS (assume)
//    Constrain the inputs to legal stimulus only.
//    Jasper's engine will only explore valid input space.
// ============================================================

   // req is a persistent request: once raised, held until gnt
   // (common arbiter protocol â€” remove if req can be pulses)
   assume_req_sticky: assume property (
      req && !gnt |-> ##1 req
   );

   // gnt is a single-cycle pulse (DUT output behavior)
   assume_gnt_pulse: assume property (
      gnt |-> ##1 !gnt
   );

// ============================================================
// 4. CORE PROPERTY (assert)
// ============================================================

   // Helper: req was seen at least once in the past 5 cycles
   // Uses a sliding window via $past
   property req_before_gnt_p;
      gnt |->
         $past(req,1) || $past(req,2) || $past(req,3) ||
         $past(req,4) || $past(req,5);
   endproperty

   // PRIMARY ASSERTION
   assert_req_before_gnt: assert property (req_before_gnt_p)
      $info("[PASS] gnt at %0t has valid req in window", $time);
      else $error("[FAIL] gnt at %0t â€” NO req in past 5 cycles!", $time);

// ============================================================
// 5. COVER PROPERTIES
//    Jasper uses these to prove the property is reachable
//    (not vacuously true). Always pair assert with cover.
// ============================================================
```

```
    // Sanity: gnt can actually fire
    cover_gnt_fires: cover property (gnt);

    // Sanity: req fires before gnt within window
    cover_req_then_gnt_1: cover property (req  ##1 gnt);
    cover_req_then_gnt_3: cover property (req  ##3 gnt);
    cover_req_then_gnt_5: cover property (req  ##5 gnt);

    // Boundary: req exactly at edge of window (t-5)
    cover_req_at_boundary: cover property (
       req ##1 !req[*4] ##1 gnt    // req at t-5, gnt at t
    );

    // Vacuity check: a case where gnt never fires
    cover_no_gnt: cover property (##10 !gnt);

// ============================================================
// 6. HELPER ASSERTIONS (debug / fault localization)
// ============================================================

    // gnt should never fire during reset
    assert_no_gnt_in_reset: assert property (
       !rst_n |-> !gnt
    );

    // If no req ever seen in 5-cycle window, gnt must not fire
    // (Redundant restatement â€” useful for Jasper witness trace)
    assert_no_spurious_gnt: assert property (
       (!$past(req,1) && !$past(req,2) && !$past(req,3) &&
        !$past(req,4) && !$past(req,5))
       |-> !gnt
    );

endmodule
```

With RAG-SVA ap_gnt_requires_prior_req1 .sv (WITH RAG)

```
//
================================================================================
// FILE:    ap_gnt_requires_prior_req.sv
// LICENSE: Personal/academic use only. Commercial use requires a Commercial License.
// DESC:    FV checker â€” gnt must be preceded by req within prior 5 cycles (SAFETY)
// GEN:     2026-05-21  |  FV MODE=YES  |  DECOMP=ATOMIC  |  MAX=16
// --------------------------------------------------------------------------
// PROPERTY INVENTORY
// --------------------------------------------------------------------------
// | Label                        | Class     | Window | Guard |
// |------------------------------|-----------|--------|-------|
// | ap_gnt_requires_prior_req      | SAFETY    |  1:5   |  N/A  |
// | mp_req_no_livelock           | FAIRNESS  |   -    |   -   |
// | cp_gnt_requires_prior_req_ant  | COVERAGE  |   -    |   -   |  Item 9
// | cp_gnt_requires_prior_req_seq  | COVERAGE  |   -    |   -   |  Item 10
// | cp_req_no_livelock_hit         | UR-5 PAIR |   -    |   -   |
// --------------------------------------------------------------------------
// UR SCAN TABLE
// --------------------------------------------------------------------------
// | mp_                  | UR-1 | UR-2 | UR-3 | UR-4 | UR-5 |
// |----------------------|------|------|------|------|------|
// | mp_req_no_livelock     | PASS | EXC  | N/A  | N/A  | PASS |
// UR-2 EXC: s_eventually global invariant â€” standalone form permitted per FV8
// --------------------------------------------------------------------------
// EXCEPTION 7b: $rose(gnt) = pure timing anchor (user confirmed)
// gnt is obligation/output â€” used only as edge-trigger reference, not as constraint
// --------------------------------------------------------------------------

module ap_gnt_requires_prior_req (
   input logic clk,
   input logic rst_n,
   input logic req,
   input logic gnt
);

  // --------------------------------------------------------------------------
  // LNG39 SEQUENCE ENDPOINT â€” historical req check
  // Clock embedded ONLY because used as .triggered endpoint
  // --------------------------------------------------------------------------
```

```
  sequence s_req_in_prior_5;
    @(posedge clk) req ##[1:5] 1;
  endsequence

  // ----------------------------------------------------------------------
  // PROPERTY DECLARATIONS
  // ----------------------------------------------------------------------

  // SAFETY â€" gnt must be preceded by req within prior 5 cycles
  // CLASS: SAFETY | LNG39 .triggered endpoint | strong() BANNED (SAFETY class, FV-SW-1)
  // EXCEPTION 7b: $rose(gnt) = pure timing anchor (user confirmed)
  property p_gnt_requires_prior_req;
    $rose(gnt) |-> s_req_in_prior_5.triggered;
  endproperty

  // FAIRNESS â€" req must eventually deassert (livelock prevention, FV8)
  // CLASS: FAIRNESS | UR-2 EXC: s_eventually global invariant
  property p_req_no_livelock;
    s_eventually !req;
  endproperty

  // ----------------------------------------------------------------------
  // ASSERTIONS
  // ----------------------------------------------------------------------

  // SAFETY
  ap_gnt_requires_prior_req: assert property (
      @(posedge clk) disable iff (!rst_n)
      p_gnt_requires_prior_req)
    else $error("[FV] ap_gnt_requires_prior_req FAIL: gnt rose at t=%0t â€" no req seen in prior 5
cycles", $time);

  // ----------------------------------------------------------------------
  // ASSUMPTIONS
  // ----------------------------------------------------------------------

  /*
  ASSUMPTION:   req must eventually deassert to prevent req livelock.
  TYPE:         Fairness/Blocking-Signal
  JUSTIFICATION: Without this constraint, FPV can construct an infinite trace
                 where req stays high permanently, creating a degenerate
```

```
            verification environment that masks real protocol violations.
 RISK LEVEL:   Low
 */
 mp_req_no_livelock: assume property (
    @(posedge clk) disable iff (!rst_n)
    p_req_no_livelock);

 // ---------------------------------------------------------------------
 // COVERAGE
 // ---------------------------------------------------------------------

 // Item 9 â€” antecedent reachability: $rose(gnt) must be reachable
 cp_gnt_requires_prior_req_ant: cover property (
    @(posedge clk) disable iff (!rst_n)
    $rose(gnt));

 // Item 10 â€” full sequence execution: req seen 1-5 cycles before gnt rises
 cp_gnt_requires_prior_req_seq: cover property (
    @(posedge clk) disable iff (!rst_n)
    req ##[1:5] $rose(gnt));

 // UR-5 PAIR â€” cp_*_hit for mp_req_no_livelock
 cp_req_no_livelock_hit: cover property (
    @(posedge clk) disable iff (!rst_n)
    !req);

endmodule
/*  [user] Add bind
[Claude]
Here is the added section: */
//
// ============================================================================
// BIND STATEMENT
//
// ============================================================================
// Replace 'my_dut' with your actual top-level DUT module name.
// Replace port connections (.clk, .rst_n, .req, .gnt) with your DUT signal names.
// This bind instantiates the checker inside the DUT hierarchy without
// modifying the DUT source â€” required for non-intrusive FV.
//
// ============================================================================
```

```
bind my_dut ap_gnt_requires_prior_req u_gnt_req_checker (
   .clk   (clk),
   .rst_n (rst_n),
   .req   (req),
   .gnt   (gnt)
);
/* What this means:

The bind statement non-intrusively attaches the checker module ap_gnt_requires_prior_req inside my_dut's scope â€” no modification to the DUT source is needed

u_gnt_req_checker is the instance name â€” you can rename it freely

Since the bind is now inline in ap_gnt_requires_prior_req.sv, you no longer need the separate bind_gnt_req.sv file

In the Questa TCL script run_questa_fv_gnt_req.tcl, remove the vlog -sv -work work bind_gnt_req.sv line â€” it is now redundant
  */
  ap_gnt_requires_prior_req.sv
```